Reinforcement Learning 1

MDPs, policies, values, TD learning, and Q-learning

Andrew Lampinen

Psych 209, Winter 2018

Introduction

Plan for these lectures

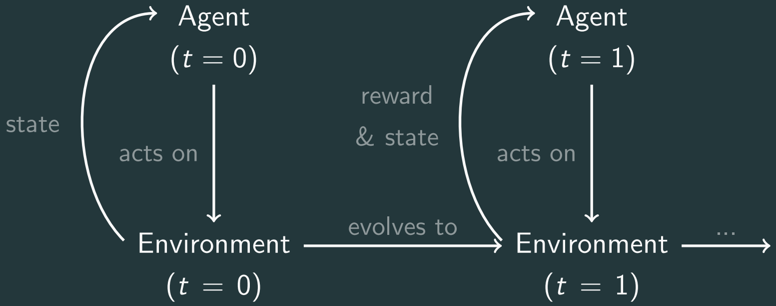
- What do the following have in common?



- ... potentially many features in common:
  - Similar structure: states, actions, occasional rewards. We'll discuss a unified formal framework for many tasks like this (**MDPs**).
  - They're not directly supervised – nobody tells you *exactly* the right answer. We'll discuss how to learn tasks like this (**Reinforcement Learning**).

Formalizing tasks: MDPs

Markov Decision Processes (MDPs)



Agents & actions

Agent:

- At each time step t, perceives the **state**, s_t decides on an **action**, a_t from the set of actions available in that state, $A(s_t)$.
- E.g. press gas, brake, turn wheel left 0.57 radians, press gas + turn right 2.2 radians, shift to 4th gear, ...

Environments, states & transitions

Environment:

- Includes other cars (and also parts of self).
- Is in a state s_t.
- After the agent takes a_t, evolves to state $s_{t+1} \in S$ according to the **transition probabilities**: $p(s_{t+1}|s_t, a_t)$.
- **Markov:** transition probabilities depend *only* on s_t, a_t, not history.

Time, & rewards

Rewards:

- Agent receives a **reward** $r_{t+1} \in R$ according to **reward probabilities**: $p(r_{t+1}|s_t, a_t, s_{t+1})$.
- E.g. fare for reaching a destination, penalty for hitting a pedestrian, ...
- **Return** is the sum of the **discounted** rewards over time: $\sum_{t=1}^{\infty} \gamma^t r_t$ for some $\gamma \in (0, 1]$.
- **Discount factor** γ tells how much we prioritize the present over the future.

$$
\begin{tikzcd}[row sep=huge, column sep=huge]
\text{Agent} \\ (t=0) \arrow[d, "\text{acts on}"'] & & \text{Agent} \\ (t=1) \arrow[d, "\text{acts on}"'] & \\
\text{Environment} \\ (t=0) \arrow[u, bend left, "\text{state}"] \arrow[rr, "\text{evolves to}"] & & \text{Environment} \\ (t=1) \arrow[u, bend left, "\text{reward \& state}"] \arrow[r, "\cdots"] & {}
\end{tikzcd}
$$

Questions?

Learning in MDPs

Policies

- How does the agent decide what to do?
- By using a **policy** π which maps states to actions.
- This policy could take many forms: picking randomly, a set of rules, a table, a neural network, or some combination thereof.
- Ideally, you would want your policy to pick the best action in every state, but what does “best” mean?

Value functions

- A natural way to define "best" states is based on expected return (which we call **value**).

$$V^{\pi}(s) = \mathbb{E}\left[\text{Return} \mid s\right] = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^t r_t \,\middle|\, s\right]$$

- Expectation because this depends on the environment's transistions and rewards, which may be random.
- But this also depends on the policy! (That's why it's V^{π}.)

Optimality

- Now we can define the **best** policy as the one that gets the **best** expected return from every state.
- We'll denote it by π^* and the associated value function by V^* or V^{π^*}.
- It is a theorem that any problem that can be specified as an MDP has at least one optimal policy.
- But how can we try to find π^* or V^*?

TD Learning and iteration

Chess

- You're playing chess against Magnus Carlsen. Let's say you estimate that you're doing pretty well, that is $V^\pi(s_t)$ is reasonably high.
- ... but then he makes a move you don't expect, and you realize you're doing worse than you thought, so $V^\pi(s_{t+1})$ is smaller.
- Can we learn from this difference?

TD Learning (applied to chess)

- Temporal Difference (TD) learning is a formal framework for learning from surprises.
- For *any* value function, we should have that

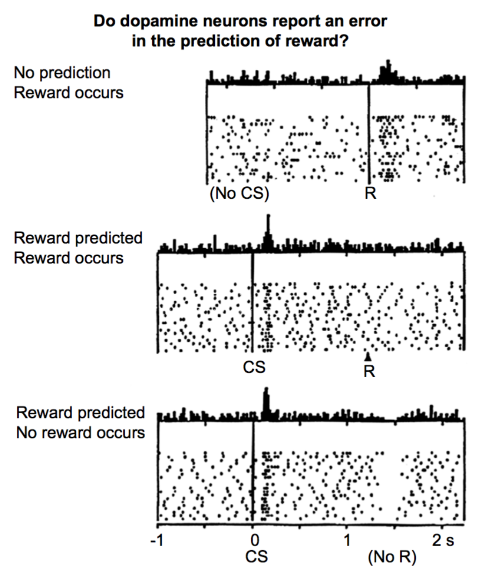
$$V^\pi(s_t) = r_{t+1} + \gamma V^\pi(s_{t+1})$$

  (This is often called the **Bellman Equation**.)
- So let's try to make this more true,

$$V^\pi(s_t) = V^\pi(s_t) + \alpha \underbrace{(r_{t+1} + \gamma V^\pi(s_{t+1}) - V(s_t))}_{\text{prediction error!}}, \quad \alpha \in [0, 1]$$

- Iterating this is guaranteed to converge to the true value function for a policy (assuming MDP is finite).

TD Learning (in the brain)



Actions

- But even if we have the optimal value function, how do we choose actions?
- In chess, it's easy – just look at the next position after you move, and figure out which one has the max value.
- But what about in more complicated situations, where we don't know how the environment will change after our actions?

Q-learning

Q-values

- The solution we'll explore incorporates the action into our values by estimating the *value of an action in a state*, which we'll denote by $Q(s, a)$:

$$Q^{\pi}(s, a) = \mathbb{E}\left[\text{Return} \mid s, a\right] = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^t r_t \,\middle|\, s, a\right]$$

- Notice that this contains the value function from before:

$$V^{\pi}(s) = \sum_{a \in A(s)} p(a|s, \pi) Q^{\pi}(s, a)$$

- But it also gives us ways of picking policies, e.g. pick the action with the highest Q.

Learning Q-values

- Using TD learning, we have another Bellman equation:

$$Q^{\pi}\left(s_t, a_t\right) = r_{t+1} + \gamma \max_{a'} Q^{\pi}\left(s_{t+1}, a'\right)$$

- So let's try to make this more true,

$$Q^{\pi}(s_t, a_t) = Q^{\pi}(s_t.a_t) + \alpha \underbrace{\left(\left[r_{t+1} + \max_{a'} \gamma Q^{\pi}(s_{t+1}, a')\right] - Q(s_t, a_t)\right)}_{\text{prediction error!}}$$

- Almost surely converges to Q^* (and by extension π^*), as long as MDP is finite and each state, action pair is visited "enough."

Questions?

Wrapping up

- Audience participation!



- Are there potential problems with what we've learned so far?